TITLE: Function Approximation Using Adaptive and Overlapping Intervals

AUTHOR(S): Rajendra B. Patil
Computer Research and Applications
CIC-3, MS B256
Los Alamos National Laboratory, Los Alamos, NM 87545

SUBMITTED TO: APIC'95 Extended Abstracts, A Supplement to the International Journal of Reliable Computing, El Paso

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
Function Approximation Using Adaptive and Overlapping Intervals

Rajendra B. Patil
Computer Research and Applications
CIC-3, MS B256
Los Alamos National Laboratory
Los Alamos, NM 87545
Function Approximation using Adaptive and Overlapping Intervals

Rajendra B. Patil

Abstract—A problem common to many disciplines is to approximate a function given only the values of the function at various points in input variable space. A method is proposed for approximating a function of several to one variable. The model takes the form of weighted averaging of overlapping basis functions defined over intervals. The number of such basis functions and their parameters (widths and centers) are automatically determined using given training data and a learning algorithm. The proposed algorithm can be seen as placing a nonuniform multidimensional grid in the input domain with overlapping cells. The non-uniformity and overlap of the cells is achieved by a learning algorithm to optimise a given objective function. This approach is motivated by the fuzzy modeling approach and a learning algorithms used for clustering and classification in pattern recognition. The basics of why and how the approach works are given. Few examples of nonlinear regression and classification are modeled. The relationship between the proposed technique, radial basis neural networks, kernel regression, probabilistic neural networks, and fuzzy modeling is explained. Finally advantages and disadvantages are discussed.

I. INTRODUCTION

A problem common to many disciplines is that of approximation of a function of n independent variables, \( z_1, \ldots, z_n \), to 1 dependent variable, \( y \), given only the value of the function at various points in the dependent variable space. This problem, in general, is of multi-variable regression. This problem and its wide variety of solutions occur in multi-variate function approximation, non-parametric regression, machine learning, neural networks, system identification, fuzzy modeling, and pattern recognition. The goal is to estimate the parameters (weights, basis functions and its parameters, coefficients, etc.) of an appropriate predetermined structure (linear or nonlinear) such that certain objective function \( \mathcal{O} \) is optimised. In case of pattern classification problems, \( \mathcal{O} \) is the number of misclassifications and in regression it is usually related to sum-of-squared error of the predicted and the actual output in dependent variable space. The task is to model the dependence of the response variable \( y \) on one or more predictor variables \( z_1, \ldots, z_n \) given the \( N \) data \( \{z_{1i}, \ldots, z_{ni}, y_i\}_{i=1}^{N} \). The unknown function that generates the data is presumed to be described by \( y = f(z_1, \ldots, z_n) + \epsilon \) over some domain \( (z_1, \ldots, z_n) \in D \subseteq \mathbb{R}_n \) containing the data. The expected value of random noise \( \epsilon \) is presumed to be zero and reflects the unknown variables that are neither controlled nor observed but affect \( y \). In pattern classification, \( y \) values are usually class labels or discrete values. In regression, \( y \) is a continuous variable. The goal then is to estimate the parameters of hyper-planes, possibly nonlinear for classification, or to estimate the regression surface from the given \( N \) data samples such that objective function \( \mathcal{O} \) (number of misclassifications or sum squared error) is minimised. In either case, the aim of function approximation is to use the data to construct a function \( \hat{f}(z_1, \ldots, z_n) \), that can serve a reasonable approximation to \( f(z_1, \ldots, z_n) \) over the domain \( D \) of interest. The approximated function \( \hat{f} \) is also expected to generalise (interpolate, extrapolate) to a certain degree over data samples in the domain \( D \) which are not part of the model building (training) process. Due to additive nature of \( \epsilon \) and demands of generalisations, wide variety of techniques are developed in many disciplines of applied mathematics, statistics, computer science and engineering.

This above problem is also known as system identification from given input-output data and is usually divided into two parts: structure identification and parameter identification. In most of the approaches, the basic structural form of \( \hat{f} \) is predetermined (linear or nonlinear) and the parameter space \( \theta \) of \( \hat{f} \) is searched to optimise \( \mathcal{O} \). The principle common to most of these approaches is the selection of...
elements from a large set of basis functions. That is, one specifies or searches for a set of basis functions \(B(\mathbf{z}, \theta)\) defined over the domain \(D\) such that most functions of \((x_1, \ldots, x_n)\) are in the span of the basis. Then in regression one uses

\[
\hat{y} = \hat{f}(\mathbf{z}) = \sum_k \alpha_k B(\mathbf{z}, \theta_k)
\]

and in pattern classification, the conditional probabilities are estimated as

\[
\hat{p}(\text{class}_j | \mathbf{z}) = \frac{1}{\sum_i \alpha_i B(\mathbf{z}, \theta_i)} \cdot \left( \sum_i \alpha_i B(\mathbf{z}, \theta_i) \right).
\]

Most current regression techniques can be expressed in this framework [4,5]. For example linear regression uses \(B(\mathbf{z}, \theta) = \{\{\theta \cdot \mathbf{z}\}\}\) which are hyper-planes. Polynomial regression uses \(B(\mathbf{z}, \theta) = \{\{\theta \cdot \mathbf{z}, \theta \cdot \mathbf{z}^2\}\}\) which are extended hyper-planes. Neural nets, non-linear regression, projection pursuit regression uses \(B(\mathbf{z}, \theta) = \{\{\theta \cdot \varphi(\mathbf{z})\}\}\) which are non-linear hyper-planes. Kernel regression, radial basis regression uses \(B(\mathbf{z}, \theta) = \{G(\theta)\}\), where

\[
G(\theta) = \exp(-\theta^2/2) \quad \text{or} \quad G(\theta) = \sqrt{2 - \theta^2}.
\]

Regression and classification trees like CART, ID3, C4, etc use \(B(\mathbf{z}, \theta) = \{1(\mathbf{z} \in R)\}\) where \(R\) is hyper-rectangle and \(\theta\) is indicator function. Here basis elements \(I(\mathbf{z}_k \in R_k)\) are selected such that \(R_k\) are disjoint. Spline regression techniques like MARS [2] uses \(B(\mathbf{z}, \theta) = \{\prod_i \pm (x_{mi} - \theta_{mi})^k\}\), here basis functions are product of a finite number of univariate splines. Fourier regression uses \(B(\mathbf{z}, \theta) = \{\sin \theta \cdot \cos \theta\}\) low and high frequency sine and cosine functions.

Various completeness theorems are proved for different set of basis functions, but it leaves an open questions as, for a given problem, what are good set of basis functions? how can a good subset of these basis functions be selected? and what algorithm is to be use to decide the parameters \(\theta\) of the chosen set of basis functions in order to optimise \(\mathcal{O}\)?

Most of the research in above regression approaches centers to answer these questions. In general there is no single approach which works best across different types of problems. The flexibility of a regression approach can be evaluated from answers to the above questions. For example the decision tree approaches like CART has discontinuous basis functions and are hyper-rectangles whose sides are parallel to coordinate axes and thus can loose accuracy and develop a large decision tree structure in modeling simple nonlinear data. MARS basis functions are continuous but bounded. On the other hand, neural network algorithms like back propagation optimises the choices over all nonlinear basis elements (hyper-planes) simultaneously [4,5].

In this paper we consider basis functions of the type \(B(\mathbf{z}, \theta) = \{\alpha(\mu(z^1 \in R), \mu(z^2 \in R), \ldots, \mu(z^n \in R))\}\), where, \(N\) is the total number of given data samples and \(R\) is a hyper-rectangle. The \(\alpha\) and \(\mu\) are indicator functions, mapping \(n\)-dimensional vector to interval \([0,1]\), i.e. \(\alpha : (\mu(\mathbf{z}))^N \rightarrow [0,1]\) and \(\mu : X^n \rightarrow [0,1]\). We use \(\mu\) as a linear function of distance from a certain point within the hyper-rectangle. This point need not be the center of the corresponding basis function. Several nonlinear choices are also possible. The \(n\)-tuple \((\mu(x_1), \ldots, \mu(x_n))\) is known as fuzzy rule in fuzzy logic [17,8]. Functions \(\mu\) and \(\alpha\) here are also known as membership function and firing strength of a fuzzy rule respectively in fuzzy set theory.

Notice that these basis functions are similar to basis functions used in a decision tree, where indicator function \(I : X^n \rightarrow \{0,1\}\), maps \(n\)-dimensional input vector to 0 or 1 while in our case, it maps to interval \([0,1]\). Another difference is that, in decision tree, adjacent basis functions (hyper-rectangles) are not overlapped, but in our case overlap is allowed and is controlled by a learning algorithm.

The basis idea is to pave the \(n\)-dimensional input space with different size overlapping hyper-rectangles such that the domain of interest is covered. For example in 2-d, the domain of interest is filled with overlapping rectangles of different sizes. In 1-d these are intervals. Using the given input-output data, each hyper-rectangle is then assigned a weighted average value \(w_i, i = 1, \ldots, m\), where \(m\) is number of hyper-rectangular basis functions. In order to decide the final output \(\hat{y}\) of the system these weighted averages are again weighted and averaged over all the basis functions paving the input domain. A simple learning algorithm is then used to adjust the sizes, centers and overlap of each hyper-rectangle in order to optimise the objective function \(\mathcal{O}\). In terms of fuzzy set theory this corresponds to searching for the parameters of a restricted class of membership functions (triangular fuzzy sets) for system identification task.

Similar approach has been used in radial basis function neural networks (RBF) [18], fuzzy system modeling [8], and number of variants kernel regression. The differences here is with type of basis functions used and the learning algorithm used to search the space of basis function parameters.

In the rest of the discussion we assume that inputs, \(x_i, i = 1, \ldots, n\) and output \(y\) is centered and normalised in the \([0,1]\) range. Rest of the discussion is outlined as follows. Section 2 details the basic concept of nonlinear interpolation using overlapping intervals. Section 3 details the search and optimisation algorithm. Section 4 describes few examples...
of classification and regression along with the performance of proposed technique. Section 5 explains the connection between fuzzy modeling, radial basis neural networks and variants of kernel regression techniques. This is followed by summary and conclusions.

II. BASIC CONCEPT

An ith interval vector of n dimension is represented as \( \mathbf{a}_i = (\mathbf{a}_{i1}, \mathbf{a}_{i2}, \ldots, \mathbf{a}_{in}) \), this defines an area (hyper-rectangle) in n-dimension, where \( \mathbf{a}_{ij} \) denotes an interval. Whenever necessary these intervals are subscripted in the usual sense for clarity. Each interval \( \mathbf{a}_i \) is specified by 3 values. \( \mathbf{l}_d, \mathbf{m}_d, \) and \( \mathbf{r}_d \) are left, in-between, and right points in an interval \( \mathbf{a}_i \) respectively, such that \( \mathbf{l}_d \leq \mathbf{m}_d \leq \mathbf{r}_d \). Notice that \( \mathbf{m}_d \) is an in-between point and need not be the center of an interval \( \mathbf{a}_i \). Function \( \mu_d(\cdot) \) is indicator function (membership function) defined over interval \( \mathbf{d} \) as

\[
\mu_d(x) = \begin{cases} 
\frac{x - \mathbf{l}_d}{\mathbf{m}_d - \mathbf{l}_d} & \text{if } \mathbf{l}_d \leq x \leq \mathbf{m}_d, \\
\frac{x - \mathbf{m}_d}{\mathbf{r}_d - \mathbf{m}_d} & \text{if } \mathbf{m}_d < x \leq \mathbf{r}_d, \\
0 & \text{otherwise.}
\end{cases}
\]  

(3)

Given a n-dimensional input vector \((x_1, \ldots, x_n)\), and interval vector \( \mathbf{a}_i = (\mathbf{a}_{i1}, \mathbf{a}_{i2}, \ldots, \mathbf{a}_{in}) \) we define

\[
\alpha_{\mathbf{a}_i} = \min\{\mu_{\mathbf{a}_{i1}}(x_1), \mu_{\mathbf{a}_{i2}}(x_2), \ldots, \mu_{\mathbf{a}_{in}}(x_n)\}.
\]  

(4)

The weighted averages \( w_i, i = 1, \ldots, m, \) of \( f(\mathbf{x}) \) over the hyper-rectangles \( \mathbf{a}_i, i = 1, \ldots, m \), is defined using the given data \( \{(x_1, y_1), \ldots, (x_m, y_m)\}^N \) as follows,

\[
w_i = \frac{\sum_{k=1}^{N} \alpha_{\mathbf{a}_i,k} \cdot y_k}{\sum_{k=1}^{N} \alpha_{\mathbf{a}_i,k}}
\]  

(5)

The final output \( \hat{y} = \hat{f}(\mathbf{x}) \) is defined as

\[
\hat{y} = \hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{m} \alpha_{\mathbf{a}_i} \cdot w_i}{\sum_{i=1}^{m} \alpha_{\mathbf{a}_i}}
\]  

(6)

In order to demonstrate how the overlapping interval basis functions using above formulation can model nonlinearity, consider following example. For simplicity consider a one dimensional function \( f(x) = y \) given only at 2 points, as \( f(0.2) = 1.0 \) and \( f(0.8) = 0.2 \). We place two intervals \( \mathbf{a} = [\mathbf{l}_a, \mathbf{m}_a, \mathbf{r}_a] = [0.0, 0.2, 0.8] \), and \( \mathbf{b} = [\mathbf{l}_b, \mathbf{m}_b, \mathbf{r}_b] = [0.2, 0.8, 1.0] \) in the domain of \( x \). This is shown in Figure 1c. along with their associated \( \mu \) functions. Notice that intervals \( \mathbf{a} \) and \( \mathbf{b} \) overlap in \([0.2, 0.8]\).

Next we calculate weighted average \( w_1 = 1.0 \) and \( w_2 = 0.2 \) of \( f(x) \) over which interval \( \mathbf{a} \) and \( \mathbf{b} \) using Equation 3, 4 and 5. The output \( \hat{y} = \hat{f}(0.2) = 1.0 \), and \( \hat{y} = \hat{f}(0.2) = 0.2 \) is calculated using Equation 6. Notice that this is the same as given \( y \) because no data points are given in \([0.2, 0.8]\). Now we show how the overlap of intervals \( \mathbf{a} \) and \( \mathbf{b} \) affects the value of \( \hat{y} = \hat{f}(x) \), \( \mathbf{m}_a \leq x \leq \mathbf{m}_b \) \((0.2 \leq x \leq 0.8)\) using above formulation. Figure 1b. shows effect of moving \( \mathbf{r}_a \), the right end point of interval \( \mathbf{a} \), towards the left from 0.8 to 0.2. Figure 1a. shows effects of moving \( \mathbf{b} \), the left end point of interval \( \mathbf{b} \) towards the right from 0.2 to 0.8. From these results it is clear that by properly placing the intervals with respect to the given data and by adjusting the centers, left and right end points of these intervals one can achieve nonlinear interpolation among given data points. Given two data points at \( \mathbf{m}_a \) and \( \mathbf{m}_b \), by placing two intervals of certain widths, the amount of overlap and the nonlinearity of associated \( \mu \) functions (membership functions) decides the degree of nonlinearity that can be modeled between the given
data points. Notice if $\mu$ functions are linear as given above (Equation 3), this can model at most second degree nonlinearities between the given two points. In order to model higher degree of nonlinearity a nonlinear $\mu$ function or more data points an correspondingly more number of basis functions at this data are needed.

Based on this simple principle, by placing sufficient number of overlapping interval basis functions paving the input domain and by adjusting their widths it is possible to approximate the unknown function from given data. The next section addresses issues related to placement of interval basis functions, and a learning algorithm for the adjustments of their parameters.

III. Regression using basis function over intervals

In this section we discuss several approaches for partitioning the input domain of interest with a set of hyper-rectangles $\mathfrak{A}_i = (\mathfrak{a}_{i,1}, \mathfrak{a}_{i,2}, \ldots, \mathfrak{a}_{i,n}), i = 1, \ldots, m$, for the function approximation task. These suggested approaches decide on number of basis functions, $n$, their initial placement in the input domain and initial widths, $\mathfrak{w}_{i,j}, \mathfrak{w}_{i,j},$ and $\mathfrak{r}_{i,j}, i = 1, \ldots, m, j = 1, \ldots, n$.

In order to model a nonlinear function underlying the given data it is then necessary to have enough basis functions placed at correct locations with appropriate overlap among them in the input domain. If all the basis functions overlap with each other completely this results in the average value of the function $f(x)$ over the domain of interest $x$. If there is no overlap there cannot be any smooth interpolation effect between data points. The appropriate locations and widths that minimised the objective function $O$ are decided upon by a simple learning algorithm. But initial values of the locations and overlap is also important. There are several possibilities to initialise the locations and widths of basis functions appropriately to reduce the burden on the learning algorithm. It is easy to notice that the most appropriate locations for basis functions are where the first derivative of underlying function is zero. This is because the nonlinearity between two adjacent points with zero first derivative can at most be of second degree and can be effectively modeled using proposed approach by simply placing two intervals with associated $\mu$ functions at each point and adjusting their widths. Moreover the initial widths for these basis functions should be inversely proportional to the degree of nonlinearity between the two adjacent points. This immediately suggest a heuristic to initialise more number of basis functions with smaller widths in the regions of input space where the function $f(x)$ has high variability and less number of basis functions with wider widths where the function has less variability. In the present context this raises the question of finding zero derivative locations of a unknown function underlying the given data. Keeping this heuristic in mind we propose several alternative for initial placement of interval basis functions and their initial widths.

In short, the problem is of placing a grid with overlapping cells of different sizes and overlap in the input domain such that, number of cells in the grid are small enough to reduce the burden on search algorithm to tune its parameters (widths), but at the same time large enough to capture the underlying nonlinearity.

A. Initialization

In the following discussion, no learning is applied to adjust the basis function parameters at this stage as the purpose here to demonstrate the sensitivity to initial placement and widths of the basis functions. Simple uniform grid:

Input space can be uniformly quantised with variable density to generate a uniform grid having non-overlapping cells. A fixed amount of overlap among these cells can be achieved by expanding each grid cell by a fixed amount. This approach does not take into account the variability of the underlying function in different regions of input space. The advantage of this approach is that the uniform density of cells can be controlled easily affecting the modeling error and decision can be made about appropriate density and overlap empirically.

Incremental grid:

In this approach at each data point we check to see if it lies in the area of basis function already placed in the domain of interest. Initially a cell is placed at the first data point with certain width, further cells are added only if the subsequent data does not lie in any of the existing cells. This approach does not take into account the variability of the underlying function in different regions of input space.

Data points as cell centers:

The limitation of being able to model nonlinearity of at most second degree between two adjacent basis functions (Equation 3) suggest a simple method of using the given data points $\{x_1, \ldots, x_n, y_1\}$ themselves as cell centers, $\{m_{d1}, \ldots, m_{dn}\}$ with $l_{k,j} = m_{k,j} + x_0$ and $r_{k,i} = m_{k,j} - x_0$, where $x_0 \in [0, 1]$. This choice of placing the interval basis functions on each data point with some overlap is very similar to kernel regression and has a disadvantage of resulting in $O(N)$ basis functions and is computationally intensive. The advantage of such
initialisation is that less number of training cycles are needed to minimise the training error.

Output variation proportional cells:

By using the estimated of variations of the output variable in different regions of the input space from the given data, the number of cells in the corresponding input regions can be made directly proportional to the variations and their widths inversely proportional to the variation. Variations of \( f(x) = y \) in different regions can be estimated using \( k \)-nearest neighbor approach, here \( k \) is a controllable parameter. Also a threshold parameter is used to detect the amount of variations. For a given value of \( k \) and threshold, regions where the \( y \) variation exceeds the threshold are paved with cells centers at all \( k \)-nearest nearest neighbors including the current data point. Regions where \( y \) variation is below the specified threshold are paved with average value of the \( k \)-nearest neighbors of the current data point. Several measures of variations of \( y \) can be used here. The simplest is the \( y_{\max} - y_{\min} \), where \( y_{\max} = \max\{y_{1\min}, \ldots, y_{N\min}\} \) and \( y_{\min} = \min\{y_{1\min}, \ldots, y_{N\min}\} \). Estimate of actual statistical variance can also be used here. In any case, these variance values are further scaled in \([\ldots]\) to standardise the adjustment of threshold parameter. This approach though more appropriate adds two extra knobs (\( k \) and threshold) to adjust the modelling process. Another simple possibility is to place cells only at two the neighbors out of \( k \) neighbors where \( y \) is maximum and minimum.

Voronoi tessellation:

This approach uses any self-organised clustering method to tessellate the input domain of interest [10]. Clustering algorithm based on vector quantisation can be used here. As in self-organisation the tessellation of input space is independent of the values of \( y \), this approach does not consider the variability in output variable \( y \).

Random placement:

Another option is to choose the placement locations randomly. Analysis shows that in very high dimensional space, the distance of a randomly selected point to its nearest neighbor will be close to maximum with very high probability. Data in high dimension is very sparse and in many situations not enough samples are available to fill up the space. In such cases the estimates of variability of output variable in different regions are not justified and not worth spending time as compared to simple random placement of the basis functions. The important conclusion is that, if a problem has many dimensions, it is not worthwhile to spend much effort to find the optimal locations for the placement of basis functions. A set of points randomly chosen are likely to be as good in the present context.

Several approaches suggested above perform well in general posing different degree of difficulty to the learning stage. The initial widths are important in the modeling process. The effect of different initial widths is shown in Figure 2. In Figure 2a, training data of 181 samples is generated using \( f(x_1, x_2) = \frac{\sin(x_1)}{x_1}, \frac{\sin(x_2)}{x_2} \). The initial cells are placed using simple uniform grid approach placing 36 initial cell centers, each at uniform spacing of 0.2. No learning is applied here at this stage as the purpose here to demonstrate the importance of initial placement and widths. Figure 2a uses initial widths of 0.1 and Figure 2b uses initial widths of 0.2.

Once the input space is partitioned in several
overlapping cells, the next stage is to tune the overlap and the centers of these cells (basis functions).

B. Learning algorithm

Proposed learning algorithm is similar to vector quantisation used for classification and clustering in pattern recognition [15]. We first explain the simple vector quantisation algorithms and then discuss the modifications used in present case.

Assume that the input domain is tessellated using number of cluster centers \( \tilde{a}_i, i = 1, \ldots, m \). These vectors are in place in the input space to approximate various parts of the input domain by quantising the input space. This algorithm is usually used for pattern classification purpose. Usually there are several clusters centers assigned to each class. The test vector \( \tilde{z} \) is decided to belong to the same class to which the nearest cluster center \( \tilde{a}_i \) belongs. Let

\[
|| \tilde{z} - \tilde{a}_i || = \min \{ || \tilde{z} - \tilde{a}_c || \}
\]

define the nearest \( \tilde{a}_i \) to \( \tilde{z} \), denoted by \( \tilde{a}_c \). The objective function is to minimise the number of misclassification and is optimised by using following rule.

Let \( \tilde{e}(t) \) be the input and \( \tilde{a}_c(t) \) represent sequences of the \( \tilde{a}_i \) in discrete time domain. Starting with some values for \( \tilde{a}_c(0), i = 1, \ldots, m \), suitably placed in the input domain, the following steps define the vector quantisation process.

\[
\tilde{a}_c(t + 1) = \begin{cases} 
\tilde{a}_c(t) + \beta(t)[\tilde{e}(t) - \tilde{a}_c(t)] & \text{if } \tilde{z} \text{ and } \tilde{a}_c \text{ of same class}, \\
\tilde{a}_c(t) - \beta(t)[\tilde{e}(t) - \tilde{a}_c(t)] & \text{if } \tilde{z} \text{ and } \tilde{a}_c \text{ of different classes}, \\
\tilde{a}_c & \text{for } i \neq c 
\end{cases}
\]

Value of \( \beta(t) \) can be constant or decrease monotonically with time. Several modifications of above basic algorithm exist, the one which is relevant to our discussion updates two cluster centers \( \tilde{a}_w \) and \( \tilde{a}_s \) that are closest to the input vector \( \tilde{z} \) instead of just the closest one [15]. One of these, \( \tilde{a}_w \), must belong to the correct class and other, \( \tilde{a}_s \), to a wrong class. Moreover, \( \tilde{z} \) must fall into the zone of values called “window”, which is defined around the midplane of \( \tilde{a}_w \) and \( \tilde{a}_s \). Assume that \( \tilde{d}_w \) and \( \tilde{d}_s \) are Euclidean distances of \( \tilde{z} \) from \( \tilde{a}_w \) and \( \tilde{a}_s \), respectively; then \( \tilde{z} \) is defined to fall in the “window” of specified width \( W \) if

\[
\min(\tilde{d}_w/\tilde{d}_s, \tilde{d}_s/\tilde{d}_w) > (1 - W)/(1 + W). 
\]

\( W \) here is a pre-specified parameter [15]. The updates of cluster center is then done as follows

\[
\tilde{a}_w(t + 1) = \begin{cases} 
\tilde{a}_w(t) - \beta(t)[\tilde{e}(t) - \tilde{a}_w(t)] & \text{if } \tilde{z} \text{ and } \tilde{a}_w \text{ of same class}, \\
\tilde{a}_w(t) + \beta(t)[\tilde{e}(t) - \tilde{a}_s(t)] & \text{if } \tilde{z} \text{ and } \tilde{a}_s \text{ of different classes}, \\
\tilde{a}_w & \text{for } i \neq c 
\end{cases}
\]

We modify the above version of vector quantisation to adjust the the widths of basis functions defined over intervals as follows. Notice that in the above vector quantisation algorithm, the “window” is defined using relative distances between the sample \( \tilde{z} \) and its two closest vectors \( \tilde{a}_w \) and \( \tilde{a}_s \) in the Euclidean sense. This defines a hyper-elliptical region around each cluster center. Parameter \( W \) controls the size of these regions. As value of \( W \) increases these regions intersect. If the given sample \( \tilde{z} \) falls in the intersecting region of these hyper-ellipses defined by its two closest neighbors, it is said to be inside the window. In the case of basis functions defined over intervals, the value of \( W \) is the same as the overlap between the two adjacent intervals. In the following, we explain how this algorithm is modified for adjusting widths of interval in our interval based regression task.

Once the input space is quantised using any one of the methods explained before, we calculate the weighted average \( w_i, i = 1, \ldots, m \) value of the underlying \( f(\tilde{z}) \) for each of these interval basis functions \( \tilde{a}_i, i = 1, \ldots, m \) using Equation 5 by going through the given data vectors. For each given vector we calculate \( \tilde{y} \) using Equation 6. Then for a given input vector \( \tilde{z} \) we first find the two closest basis functions \( \tilde{a}_w \) and \( \tilde{a}_s \) as follows. We first calculate \( \tilde{a}_i, i = 1, \ldots, m \) for all basis functions using Equation 4. We choose the two basis functions \( \tilde{a}_w \) and \( \tilde{a}_s \) having the largest and the second largest \( \alpha \) values. Let the previously calculated weighted average for these two closest basis functions be \( w_w \) and \( w_s \) respectively. If the given input vector \( \tilde{z} \) falls in the overlapping region of the two basis functions \( \tilde{a}_w \) and \( \tilde{a}_s \), then we update the spreads of interval components of \( \tilde{a}_i \) basis functions using the modified vector quantisation algorithm as follows. We first find the jth appropriate interval component \( \tilde{a}_{i,j} \) of \( \tilde{a}_w \) to be updated as

\[
\mu_{i,j}(\tilde{z}) = \min \{ \mu_{i,j}(x_1) \}
\]

This is that interval component of \( \tilde{a}_w \) which has lowest \( \mu \) value for the corresponding component of input vector \( \tilde{z} \). Once this interval component \( \tilde{a}_{i,j} \) is found, its width is adjusted as follows.

Notice that \( \tilde{a}_{i,j} = [\tilde{a}_{i,j}, m\tilde{a}_{i,j}, r\tilde{a}_{i,j}] \) and \( \tilde{a}_{w,j} = [\tilde{a}_{w,j}, m\tilde{a}_{w,j}, r\tilde{a}_{w,j}] \). This jth interval component is adjusted as follows. The process is shown in Figure B. If the jth component of input vector \( \tilde{z}, x_j \), falls on the right side of \( m\tilde{a}_{w,j} \), i.e. \( x_j > m\tilde{a}_{w,j} \) and if \( (y - \hat{y})(w_w - w_s) > 0 \), the \( r\tilde{a}_{w,j} \) is moved further away from \( m\tilde{a}_{w,j} \) towards \( m\tilde{a}_{w,j} \).

If the jth component of input vector \( \tilde{z}, x_j \), falls on the left side of \( m\tilde{a}_{w,j} \), i.e. \( x_j < m\tilde{a}_{w,j} \) and if \( (y - \hat{y})(w_w - w_s) > 0 \), the \( l\tilde{a}_{w,j} \) is moved further...
away from \( m_{a,j} \) towards \( m_{a,j} \).

If the \( j \)th component of input vector \( \vec{z}, z_j \), falls on the right side of \( m_{a,j} \), i.e. \( z_j > m_{a,j} \) and if \( (y - \hat{y})(w_u - w_v) < 0 \), the \( a_{r,j} \) is moved closer to \( m_{a,j} \) towards \( w_u \).

If the \( j \)th component of input vector \( \vec{z}, z_j \), falls on the left side of \( m_{a,j} \), i.e. \( z_j < m_{a,j} \) and if \( (y - \hat{y})(w_u - w_v) < 0 \), the \( a_{l,j} \) is moved closer to \( m_{a,j} \) towards \( w_u \).

In cases where only one basis function \( \alpha_u \) has non-zero value, \( \alpha_u > 0 \) and \( \alpha_u = 0 \) implies that the input vector falls outside of window \( W \). Then the center \( m_{a,j} \) is adjusted by moving it towards the current input vector \( \vec{z} \). The weighted average \( w_i, i = 1, \ldots, m \) are also adjusted in each cycle. The above algorithm is express more clearly in following pseudo code.

\[
\text{if } (\alpha_u > 0) \text{ then}
\quad \text{if } (z_j > m_{a,j}) \text{ then}
\quad \quad r_{a,j}(t + 1) = r_{a,j}(t) + \beta(t)[m_{a,j} - r_{a,j}(t)]
\quad \text{else}
\quad \quad r_{a,j}(t + 1) = r_{a,j}(t) + \beta(t)[l_{a,j} - r_{a,j}(t)]
\quad \text{else}
\quad \quad \text{if } (y - \hat{y})(w_u - w_v) > 0 \text{ then}
\quad \quad \quad l_{a,j}(t + 1) = l_{a,j}(t) + \beta(t)[m_{a,j} - l_{a,j}(t)]
\quad \quad \text{else}
\quad \quad \quad l_{a,j}(t + 1) = l_{a,j}(t) + \beta(t)[r_{a,j} - l_{a,j}(t)]
\quad \text{else}
\quad \quad \quad m_{a,j}(t + 1) = m_{a,j}(t) + \beta(t)[\vec{z} - m_{a,j}(t)]
\quad \text{else}
\quad \quad \quad w_u(t + 1) = w_u(t) + \gamma(t)\alpha_u(t)[y - \hat{y}].
\]

IV. Examples

In this section we demonstrate the proposed regression technique using examples from nonlinear multivariable regression and pattern classification.

A. Regression

Example 1:

The function

\[
f(x_1, x_2) = \frac{\sin(x_1) \sin(x_2)}{x_1 \cdot x_2}
\]

was sampled uniformly generating 169 \( \{x_1, x_2, y\} \) training sample as shown in Figure 3a. Initial placement of the cell partitioning input domain was done using the output variation proportional cells approach leading to 45 initial cells. Threshold value was set to 0.5. These initial placements are shown in Figure 3b along with the contour map of \( f(\vec{z}) \). Notice that, more number of initial cells are allocated where the function has high variations and less where it has less variations. Each cell width was set to the maximum distance from the \( k_n = 8 \) distances for both the components. A test set consisting of 1445 sample was created using the same function. Figure 3c shows the results over test data after 1 cycle of training. Figure 3d shows the results over test data after 20 cycles of learning. The mean squared error was 0.0021.

Example 2:

The function

\[
f(x_1, x_2) = x_1 \exp(-x_1^2 - x_2^2)
\]

was sampled uniformly generating 81 \( \{x_1, x_2, y\} \) training sample as shown in Figure 4a. Initial placement if the cell paving input domain was done using the output variation proportional cells approach leading to 18 cells. Threshold value was set to 0.5. These initial placements are shown in Figure 4b along with the contour map of \( f(\vec{z}) \). Each cell width was set to the maximum distance from the \( k_n = 8 \) distances for both the components. A test set consisting of 1681 sample was created using the same function. Figure 4c shows the results over test data after 1 cycle of training. Figure 4d shows the results over test data after 20 cycles of learning. The mean squared error was 0.0018.

B. Classification

In general the problem of pattern classification [9] can be considered as a regression problem by applying threshold to the predicted output of the regression model. Here we consider two well known instances of pattern classification problems from machine learning and analyse the accuracy of proposed technique.

Example 1:

In this example the dataset used is from the UCI machine learning database called "Wisconsin breast cancer database" [6]. There are 9 input attributes, all discrete on a scale 1 - 10 (integer) and 1 binary output attribute (0=benign, 1=malignant). This encoding is used directly, except that the values are scaled to a 0...1 range and that the output is represented with 0 or 1. There are 16 missing values for attribute 6; they are encoded as 0.3 since the average value of that attribute is roughly 3.5. Class distribution is, benign cases consist of 66% and malign cases consist of 33% of the total 699 samples. The dataset was divided into two parts, training data consisted of 350 sample out of 699 samples and testing data consisted of 349 samples. Using 350 training data and 50 random initial cells, each with width of 0.9 was used to initialise the model and trained for 10 cycles. The trained model was then tested on the testing data. A threshold of 0.5 was used to infer be-
nign (if $\hat{y} > 0.5$) and malignant (if $\hat{y} \leq 0.5$) cancer cases leading to 120 correct and 4 incorrect classifications out of true 124 malignant cases (96.7% correct) and 210 correct and 15 incorrect classifications out of 225 true benign cancer cases (93.3% correct). The average classification accuracy was 95%, which is comparable to the best known classification accuracy of 96% on this data set.

Another attempt at the same problem with different method of initial 49 cells using self-organising vector quantisation method resulted in, for class 1, 213 correct, 12 incorrect (94.6% accuracy), and class 2, 120 correct and 4 incorrect (96.7% accuracy) classifications in 10 cycles with 0.9 initial widths for all 49 cells.

**Example 2:**
This example considers pattern classification dataset related to heart disease from UCI repository [6]. This database contains 13 attributes (which have been extracted from a larger set of 75). Attribute information consists of age, sex, chest pain type (4 values), resting blood pressure, serum cholesterol in mg/dl, fasting blood sugar > 120 mg/dl, resting electrocardiographic results (values 0,1,2), maximum heart rate achieved, exercise induced angina, oldpeak = ST depression induced by exercise relative to rest, the slope of the peak exercise ST segment, number of major vessels (0-3) colored by fluoroscopy, thal: 3 = normal; 6 = fixed defect; 7 = reversible defect. Variable to be predicted is, absence (0) or presence (1) of a heart disease. The best results reported are 77% accuracy in predicting the classes [6]. In our attempt, out of 270 available samples, 135 sample were used for training and 135 for testing. In our model we used 50 random initial cells, each with initial width of 0.9. After 3 cycles of training, out of 60 positive cases 53 were predicted correctly (88%) and out of 75 negative cases 62 were predicted correctly (82%) over the test data set.

V. Connections with other approaches

In this section we discuss the issues related to conceptual similarities between the approaches from different areas. The concept of partitioning the input space using set of basis functions chosen from a particular family is common to fuzzy modeling, radial basis functions networks, decision trees, and kernel regression. In general, the process of data modeling is divided into two step in each of these approaches. The first step is the approximate partitioning of input space and associating local functions (parametric or non-parametric) to these partitions. The second step is the inferencing process using information within all or specific partitions and their associated functions. Though the same concept is used here across all these approaches, the methods to achieve the goal are different and much can be learned across these techniques creating new hybrid and flexible approaches. For example learning capabilities of radial basis networks can be used in fuzzy modeling, or the partitioning stage in fuzzy modeling can use decision tree concepts.

In order to keep the following discussion at conceptual level we consider partitioning and inferencing as separate stages. Partitioning stage can be static as in conventional fuzzy modeling where set of rules are given or in kernel regression where the type and parameters of kernel used is fixed. It could be subjective or objective. In a dynamic par-
**Figure 4:** (a) Training data of function \( f(x_1, x_2) = \frac{\sin(x_1) \cdot \sin(x_2)}{x_1 \cdot x_2} \), (b) Contour map of \( f(x) \) (dotted) superimposed with initial cell placement (+) using output variation proportional cells approach. The points where function is sampled are also shown (.), (c) Results of function approximation after 1 cycle of learning and (d) Results of function approximating after 20 cycles of learning the la, ma, ra, values of initial cells. The mean squared error was 0.0021.

Partitioning case like radial basis function networks, or the approach proposed in this paper, initial partitioning is augmented with a modification procedure to tune certain parameters reflecting changes in the shape of the partitions. Partitioning process also associates local functions to each partition and these local functions, (called basis functions in general) allows a unified analysis of these techniques. One can associate simple functions as a constant, polynomial, parametric function, or complex non-parametric functions to these individual partitions to achieve the task of data modeling.

The inferencing process can be viewed as testing these approaches for new cases. A form of “weighted averages” over the partitions and associated functions is a common inferencing procedure across these approaches.

**Partitioning:**
Kernel regression (similarly probabilistic neural networks) uses \( \{B(z, \theta)\} = \{G(\theta)\} \), where \( G(\theta) = \exp(-\theta^2/2) \) or \( G(\theta) = \sqrt{\pi^3 + \theta^2} \). Each given data point is used as a center of these kernels. This effectively partitions the input space. If the kernels are Gaussian the partitioning of input space is done using Gaussian footprints. Depending on the density of the given data and width of the kernels these partitions can be overlapping or disjoint. Each data point and associated kernel here is a basis function. Radial basis function networks uses similar
approach but usually do not used the whole data as kernel centers.

Regression and classification trees like CART, ID3, C4, etc use \( \{B(z, \theta)\} = \{I(z \in R)\} \), where \( R \) is hyper-rectangle and \( I \) is indicator function. Here basis elements \( I(z \in R_k) \) are selected such that \( R_k \) are disjoint. This partitions the input space into disjoint hyper-rectangles whose axes are parallel to the coordinates of the input space. The function associated with each partition is usually a constant (decision tree) or a polynomial (regression tree), or in a more complicated case a complex non-parametric model. Each hyper-rectangle with associated function here is a basis function.

Fuzzy modeling uses basis functions of type \( \{B(z, \theta)\} = \{\alpha(\mu(z \in R))\} \), where \( R \) is a hyper-rectangle, \( \alpha \) and \( \mu \) are indicator functions, mapping \( n \)-dimensional vector to interval \([0,1]\), i.e. \( \alpha : (\mu(z))^{n} \to [0,1] \), and \( \mu : X^{n} \to [0,1] \). Functions \( \mu \) and \( \alpha \) here are know as membership function and firing strength of a fuzzy rule respectively in fuzzy set theory. The \( n \)-tuple \( (\mu(z_1),...,\mu(z_n)) \) is know as fuzzy rule in fuzzy logic. A fuzzy system is defined using a set of fuzzy rules. Usually from the knowledge of the underlying system these rules are defined subjectively and are later tuned. These set of rules cover the complete dynamic range of the inputs. Each rule defines a partition in the input space. The function associated with each partitions is decided upon by the function \( \mu \) in that part of the

Figure 5: (a) Training data of function \( f(z_1,z_2) = z_1 \exp(-z_1^2 - z_2^2) \) (b) Contour map of \( f(z) \) (dotted) superimposed with initial cell placement (+) using output variation proportional cells approach. The points where function is sampled are also shown (.), (c) Results of function approximation after 1 cycle of learning and (d) Results of function approximation after 20 cycles of learning the \( \hat{\mu}, \hat{m}, \hat{v}, \hat{a} \) values of initial cells. The mean squared error was 0.0018.
input space. Each rule and associated \( \mu \) function is a basis function.

The central idea across all these approaches is then to place certain types of basis functions in the input space. Notice that the placement of these basis functions is subjective or objective. Depending on the approach the subjective choices can be aided with process of learning the initial locations and parameters of the initial basis functions. Next we discuss the similarity in the inferencing process across these approaches.

**Inferencing:**

Inferencing mechanism across above approaches can be, in general, characterised as "sophisticated weighted averages" or "sophisticated weighted sum" over the basis functions that partition the input space. In essence, this makes the inferencing process as a lookup table with "sophisticated weighted averages". In kernel regression, for a new sample, inferencing is done by integrating response over basis functions in the neighboring regions of partition where the new case falls. In radial basis function networks, new sample is inferred upon by taking normalised response over all basis functions in the input space. In conventional decision trees, no weighting or averaging is done but a constant response is produced. In regression trees the type of parametric or non-parametric basis functions associated with each partition decided what kind of "sophisticated weighted average" is produced as a response. In fuzzy logic many variations of weighted averages are used to combine responses from all the basis functions (rules) in the input domain.

In general, fuzzy rules in fuzzy modelling are same as the basis functions of radial basis function network, kernels in kernel regression, and leaves of a tree structure in decision tree or regression tree.

In summary, under minor restriction and mathematical frameworks under which above approaches are developed and researched, it is clear that they use same concept to achieve the same goal of data modeling. The advantage of this broad view among these approach is that, we can apply advances and new developments of one approach to other and vice versa. One such attempt is explained in this paper and can be viewed as applying learning capabilities of vector quantisation in learning the parameters of basis functions (membership functions) of a fuzzy system.

**VI. Summary and Conclusions**

In this paper we have proposed one simple method to solve problem of multivariate regression common to wide variety of disciplines. Our approach uses linear basis functions defined over intervals. A learning algorithms is proposed to tune locations and widths of these basis functions to minimise an objective function. Explanation of how any why such approach works is given with a simple example. Utility of the proposed method is demonstrated with two problems of nonlinear regression with two input and one output variables. Performance over two well known pattern classifications problems of high dimension is also reported. We also explain the similarities of proposed approach with existing techniques of radial basis function networks, kernel regression, decision trees, and fuzzy modeling at a conceptual level. The concepts of partitioning input space with basis functions and an inferencing mechanism over new cases once the model is built are very similar across these techniques. This similarity can be used to create hybrid and more flexible approaches to the data modeling problem.

Issues related to the initial placement of basis functions and the type of basis functions is important. With the proposed basis functions a wide variety of initialisation techniques work well with a different degree of difficulty in the learning stage. It is possible to use partitioning techniques used by decision trees to decide on initial placement problem. Also the proposed learning algorithm can be used in decision trees to make them adaptive. The parametric nature of basis functions used in local regions of the input space can be made arbitrary complex and wide variety of learning algorithms can be used to tune their parameters. Proposed method can also be used in an adaptive fuzzy rule base system where rules (basis functions) are learned from the available input-output data.

**References**


